Machine Learning with Graphs (MLG)

Homework 2 Report

吳岱桀 E14051350

1 INTRODUCTION

In the world nowadays, a new data type ‘graph’ has proposed to formulate interaction among entities. Take stream platform like Netflix or Spotify for example, we may have the history of user interaction which is the record that user may watch or listen to certain movies or music, and we can convert the history data into user-item graph for the use of recommending system. Other applications such as friend recommendation, atomic connection in Chemistry and citation relationship in are also common aspect to make use of the graph.

One of major problem in the graph data that we are curious is *link prediction*. Link prediction is to predict whether two nodes in a network are likely to have a link(edge). In in a social network graph, we may want to know if there exist friendship links among users, while interactions between genes and proteins can be formulated as a link prediction problem in biological network graph.

With the network in lots of application has become more complicated, the amount of node and edge(link) have become more and more, and it thus cause the efficiency problem to deal with link prediction issue in the large-scale graph. In this project, we have tried a variety of methods such as heuristic feature extraction, representational learning approach to obtain node embedding and even graph neural network method like graph autoencoder.

In the following sections, we will first illustrate how we conduct different type of feature engineering method to obtain the useful representation of each node. After feature engineering part, we’ll explain how to model the link prediction problem with the feature of a node pair. Different models have been trained based on different sets of features to investigate the relationship between input features and the performance of prediction.

2 METHODOLOGY

2.1 Problem Formulation

The target of this project is to predict the link probability of a node pair. Link prediction approaches for this setting learn a model that maps the links in the graph to a real value between 0 and 1 representing the probability.

- *Vertex set* is $V = \{1, 2, \ldots, x, y, \ldots, N\}$ where x and y are some generic vertices of interest
- *Edge set* is $E = \{(x, y) : x, y \in V\}$
- *Graph* G is defined by a pair (V, E)
- An *adjacency matrix* of G is a matrix $A = [a_{xy}]_{N \times N}$ such that $a_{xy} = 1$ if $(x, y) \in E$ otherwise $a_{xy} = 0$.
- A *set of neighbors* of vertex x in graph G is $\Gamma(x)$, namely $\Gamma(x) = \{y : (x, y) \in E\}$.
- k_x is the *degree* of vertex x in graph G
- *Proximity measure* s_{xy} is a function $s_{xy} : G, x, y \mapsto \Re$.

2.2 Direct Feature Engineering and Modeling

In this part, we'll introduce the heuristic methods to get the score value set in a node pair based on different characteristic. After the calculation of the proposed score set, we'll train models such as SVM, XGBoost and MLP to compare the performance. The following section are the methods to get the score.

2.2.1 Preferrential Attachment

It is a measure to indicate that new links will be more likely to connect higher-degree nodes than lower ones. The score to compute Preferrential Attachment of a node pair is:

$$s_{xy} = k_x \cdot k_y.$$

2.2.2 Common Neighbors

Two nodes are more likely to be connected if they are connected to the same set of other nodes. It was used in the study of collaboration networks, showing positive relation between number of common neighbors and probability of collaborating in the future. The score to compute Common Neighbors of a node pair is:

$$s_{xy} = |\Gamma(x) \cap \Gamma(y)|$$

2.2.3 Jaccard Index

It measures the proportion of common neighbors in the total number of neighbors. It's similar to the concept as Common Neighbors, but it is a normalized form of common neighbors. The score to compute Jaccard Index of a node pair is:

$$s_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

2.2.4 Adamic–Adar Index

It measures similarity between two nodes by weighting fewer neighbors more heavily, which counts common neighbors by assigning weights to nodes inversely proportional to their degrees. That means that a common neighbor, which is unique for a few nodes only, is more important than a hub. The score to compute Adamic–Adar Index of a node pair is:

$$s_{xy} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log k_z}$$

2.2.5 Resource Allocation

This measure is motivated by a resource allocation process. It measures how much resource is transmitted between x and y. It's similar to Adamic–Adar Index but it punishes the high-degree common neighbors more heavily than Adamic–Adar Index. The score to compute Resource allocation of a node pair is:

$$s_{xy} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{k_z}$$

2.2.6 Katz Index

It considers all paths between two nodes and weighs shorter ones more heavily. The score to compute Katz Index of a node pair is:

$$s_{xy} = \sum_{l=1}^{\infty} \beta^l |paths_{xy}^{<l>}|$$

β is a free parameter. The sum converges when β is lower than the reciprocal of the largest eigenvalue of adjacency matrix.

2.2.7 Salton Index (cosine similarity)

It measures the cosine of the angle between columns of the adjacency matrix, corresponding to given nodes. The score to compute Salton Index of a node pair is:

$$s_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{\sqrt{k_x \times k_y}}$$

2.2.8 Sørensen Index

It is similar to the Jaccard Index as it measures the relative size of an intersection of neighbors' sets. The score to compute Sørensen Index of a node pair is:

$$s_{xy} = \frac{2|\Gamma(x) \cap \Gamma(y)|}{k_x + k_y}$$

2.2.9 Hub Promoted Index

It assigns higher scores to links adjacent to hubs (high-degree nodes), as the denominator depends on the minimum of the degrees of the nodes of interest. The score to compute Hub Promoted Index of a node pair is:

$$s_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{\min\{k_x, k_y\}}$$

2.2.10 Hub Depressed Index

In contrast to Hub Promoted Index, it assigns lower scores to links adjacent to hubs. It penalizes large neighborhoods. The score to compute Hub Depressed Index of a node pair is:

$$s_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{\max\{k_x, k_y\}}$$

2.2.11 Leicht-Holme-Newman Index

It is a variant of Common Neighbors, similar to Salton Index. The score to compute Leicht-Holme-Newman Index of a node pair is:

$$s_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{k_x \cdot k_y}$$

2.2.12 Average Commute Time

To compute Commute Time, calculation of m(x,y) is needed. m(x,y) is the average number of steps required by a random walker starting from x to reach y. Sum of two directional commute times is taken to achieve symmetrical measure. Hence two nodes are similar if they are closer to each other and have smaller commute time, reciprocal is needed. The score to compute Average Commute Time of a node pair is:

$$s_{xy} = \frac{1}{n(x,y)} = \frac{1}{m(x,y) + m(y,x)}$$

2.2.13 Modeling

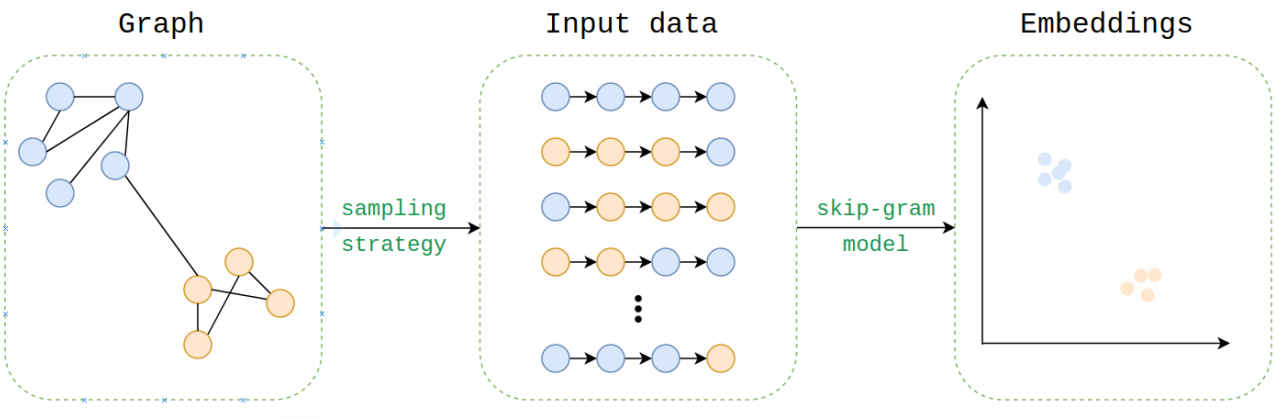
As baseline model, the input feature of this section are the 12 scores mentioned above. Though feature of each of node is provided, it's too sparse and the dimension of original node feature is too high. After considering the balance between the advantage of add original node feature and the disadvantage of sparsity problem, we decide not to concatenate the original node feature with the score set calculated above because the score features are already informative and the sparsity issue brought by concatenation operation would make it harder for the model to find decision boundary.

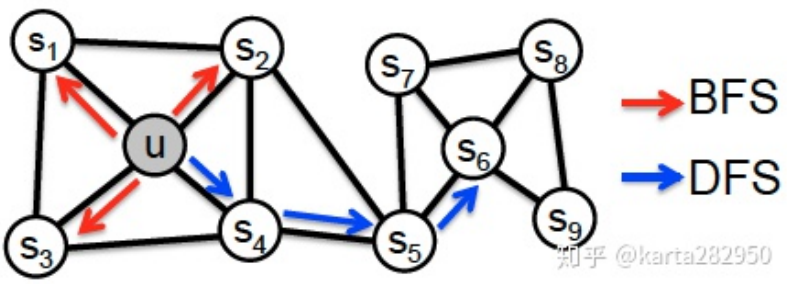
2.3 Node Embedding Method

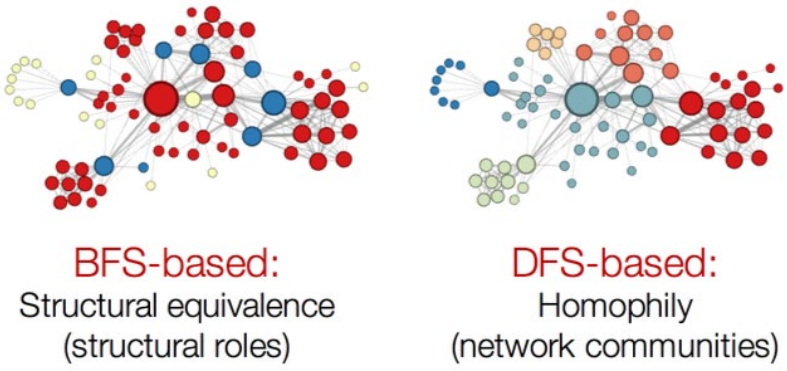
Different from directly calculating the score between a pair of nodes, embedding method makes use of homophily and structural property to learn a low-dimension feature representation of each node. To predict status of linking, we consider taking concatenation and inner-product operation on the embedding of node pair.

2.3.1 node2vec

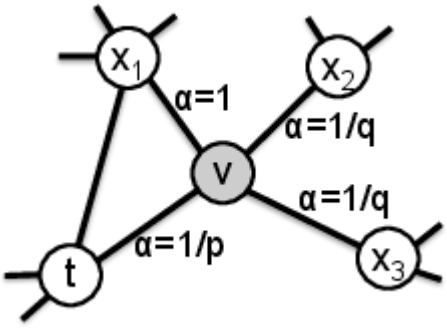
Inspired by the Word2Vec in NLP application and Deep Walk algorithm, node2vec is a unsupervised learning method which generates random walks from each node of the graph.





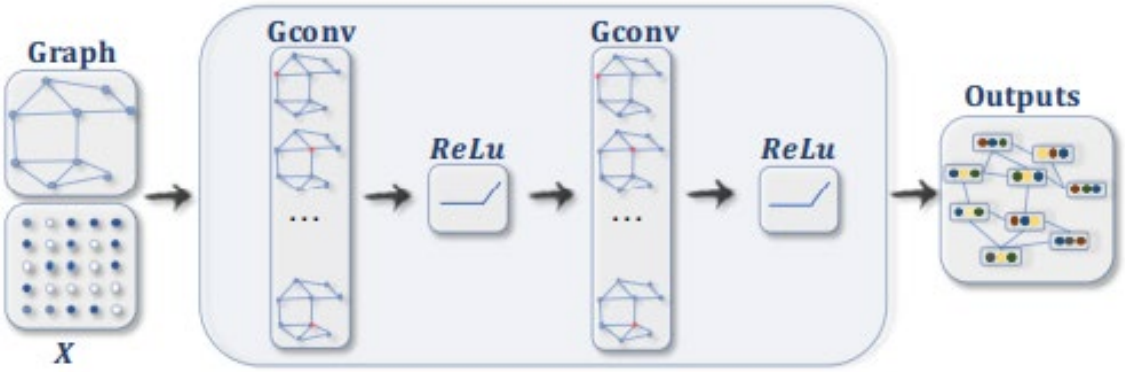


Combine Breadth First Search and Depth First Search with random walk sequence generation methods, node2vec is based on the measurement of homophily similarity and structural similarity, extracting structural low-dimensional features. There're four major hyperparameters in Node2vec's sampling strategy, which are Number of walks, Walk length, P(Return hyperparameter) and Q(Inout hyperaprameter).

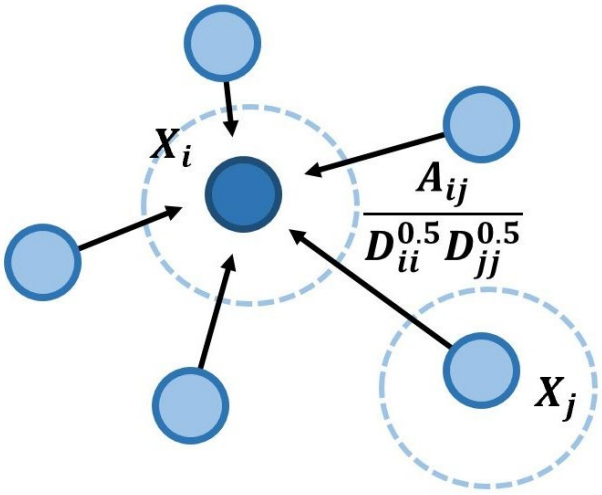


Number of walks decides number of random walks to be generated from each node in the graph, while walk length affect the amount of node it would be in each random walk. The figure above can illustrate the role of P and Q. After transitioning to node v from t, the return hyperparameter P, and the inout hyperparameter Q, control the probability of a walk staying inward revisiting nodes (t), staying close to the preceeding nodes (x1), or moving outward farther away (x2, x3).

2.3.2 Graph Convolutional Network（GCN）



In recent years, Graph Neural Network has been proposed to utilize the advantage of neural network on graph-related application. One of important network architecture is Graph Convolutional Network. As Convolutional Neural Network has been commonly used in Computer Vision field, CNN is based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps. GCNs perform similar operations where the model learns the features by inspecting neighboring nodes. The major difference between CNNs and GNNs is that CNNs are specially built to operate on regular (Euclidean) structured data, while GNNs are the generalized version of CNNs where the numbers of nodes connections vary and the nodes are unordered (irregular on non-Euclidean structured data).



$$
\begin{aligned}
aggregate(X_i) &= \hat{D}^{-0.5}\hat{A}\hat{D}^{-0.5}X \\
&= \sum_{k=1}^{N} \hat{D}_{ik}^{-0.5} \sum_{j=1}^{N} \hat{A}_{ij} X_j \sum_{l=1}^{N} \hat{D}_{il}^{-0.5} \\
&= \sum_{j=1}^{N} \hat{D}_{ii}^{-0.5} \hat{A}_{ij} X_j \hat{D}_{jj}^{-0.5} \\
&= \sum_{j=1}^{N} \frac{1}{\hat{D}ii^{0.5}} \hat{A}_{ij} \frac{1}{\hat{D}_{jj}^{0.5}} X_j \\
&= \sum_{j=1}^{N} \frac{\hat{A}_{ij}}{\sqrt{\hat{D}ii\hat{D}jj}} X_j
\end{aligned}
$$

In GCN, node embedding can be learned during the process of ‘aggregating’ the information of neighbor node. By means of aggregation, it applies summation or

mean function on a node's neighbor information. The node embedding relationship between two layers can be expressed as the following formula:

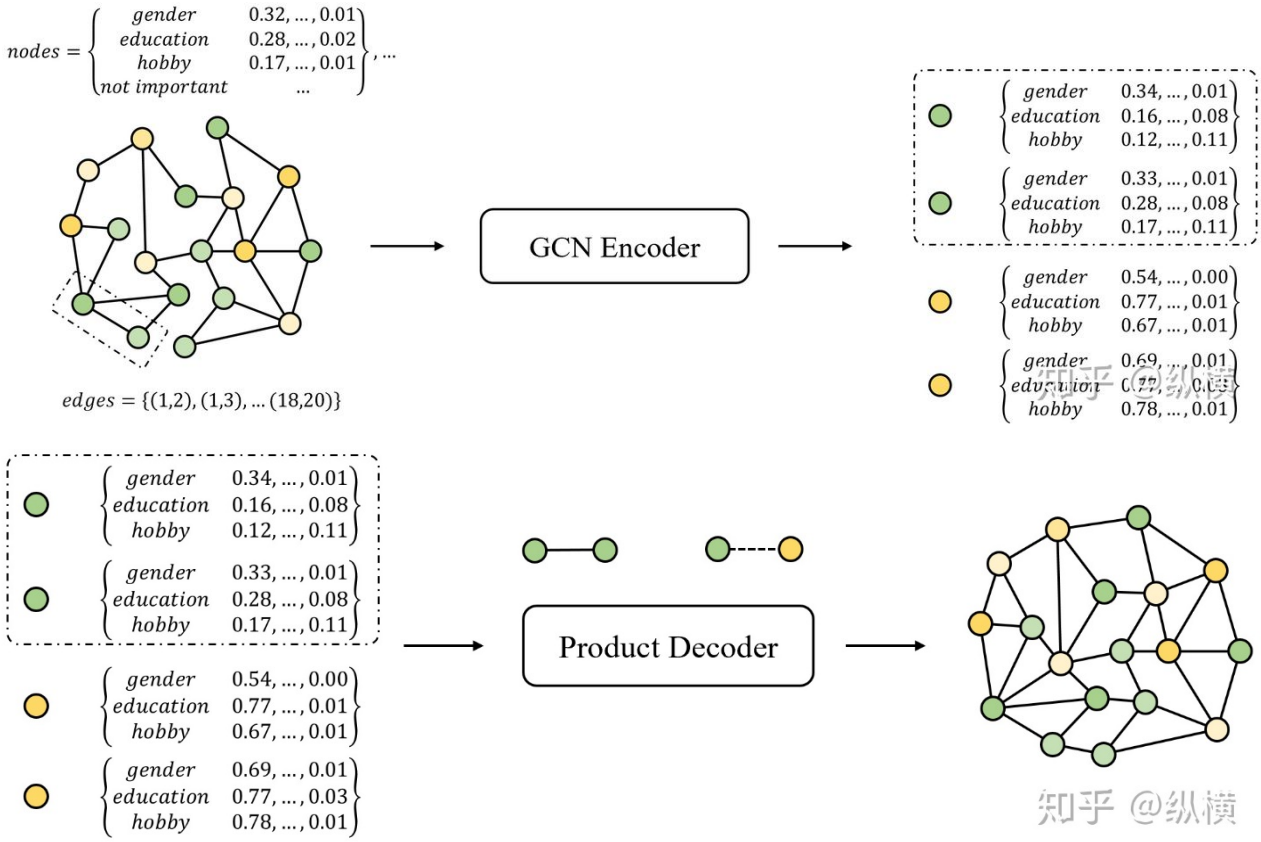
$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

H denotes the node embedding at different layer. D is degree matrix of nodes, while W is learnable weights.

2.3.3 Modeling

The node embedding generated by node2vec can be used as input feature for Machin Learning model. For example, we may take concatenation, summation or inner product operation on node embedding of a node pair and treat the processed feature vector as the input to the model like SVM, Random Forest or other tree-based models.

Unlike node2vec that two-stage training is needed(first training the node2vec to obtain node embedding and training machine learning model to take node embedding as input), GNN-based method is generally seen as an end-to-end approach.



The link prediction problem can be formulated as a reconstruction problem that attempts to reconstruct the link status in a whole graph by the help of node feature and partial existing links in the graph. To achieve the reconstruction target, the architecture of Graph Auto Encoder(GAE) is proved be a effective method. A simple GAE is composed of two part as illustrated in the above figure. The encoder part is a multi-layer GCN that yields the node embedding. As for the decoder, it takes inner-product operation on node embedding of a node pair. Cross Entropy is used as loss function to optimize the prediction task that output the probability of link status in the graph.

One of advantages of GNN is that it is able to take node feature into consideration, which the structure information of the graph and node feature are utilized together to get the node embedding, while node2vec can not take the advantage of node feature.

Though sparsity problem of node feature, we believe it still helps the model to get better decision boundary. As a result, we propose to train a MLP autoencoder to get a low-dimensional node feature embedding and then the processed node feature embedding can be seen as extra feature to both node2vec and GAE methods.

2.4 Performance Evaluation

To measure the performance of our trained model, we utilized AUC (area under curve) on ROC curve to measure the relationship between true positive rate and false positive rate in different classification threshold. In addition to AUC of ROC curve, we also used precision to measure how much percentage of the predicted result is true (true positive / predicted positive).

3 EXPERIMENT & ANALYSIS

In order to demonstrate the effect of different feature engineering and modeling process, we would only show the result of Dataset 1 in the report. The reset of the dataset show the same improvement trend

3.1 Model Using Score Feature(baseline)

In this section, we'll calculate 12 scores on both positive(already existing in the graph) and negative links(negative sampling) as input feature. The modelS used in this section include SVM, Logistic Regression, Random Forest and LightGBM. However, there's no significant improvement in different model. The best AUC is

0.5752 and Precision is 0.6315. The initial thought on the poor performance is that it fails to successfully express the homophily and structural property by merely 12 score feature.

3.2 Model Using node2vec

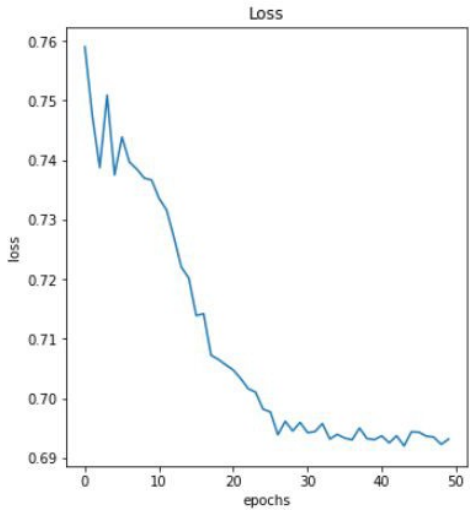
The node embedding generated by node2vec can be directly used in most of machine learning model. We confirm that performance difference is more significant in the change of operation than the change in model. The result to compare the summation and concatenation operation is in the following table with the node dimension 100

	Summation	Concatenation
AUC	0.8117	0.7804
Precision	0.8836	0.8651

To combine the node feature and node embedding of each of node together, we obtain low-dimensional node feature embedding by autoencoder and then concatenate node feature embedding and node embedding as input feature for model. The best AUC is 0.8562 and Precision is 0.8918.

As we can see in the result above, the overall performance improves with the help of node2vec. It verifies that the great ability of node2vec to represent the structure of graph. Besides, the we find that the node feature is confirmed helpful after the reduction in dimensionality.

3.3 Model Using GAE



We have done experiment on using only node feature embedding and concatenation of node feature embedding and node embedding of node2vec as input of GAE. The encoder is composed of 3 layer GCN and decoder is a inner-product decoder.

	node feature embedding	concatenation
AUC	0.8341	0.8495
Precision	0.9005	0.9089

4 CONCLUSION

In this project, we have tried different methods to solve link prediction problem. It shows that the combination of node2vec and GNN is better than the single GNN setting. We can also learn that both structure of graph and node feature are essential for link prediction problem.

5 REFERENCE

[1] Proximity-based Methods for Link Prediction (r-project.org)

[2] node2vec (stanford.edu)

[3] node2vec: Scalable Feature Learning for Networks - 知乎 (zhihu.com)

[4] 如何理解 Graph Convolutional Network（GCN）？ - 知乎 (zhihu.com)

[5] 如何理解链接预测（link prediction）？ - 知乎 (zhihu.com)